Лабораторна робота №11

Програмування команд SSE у модулях на асемблері

Мета: Навчитися програмувати модулі на асемблері, у яких містяться команди SSE, команди x87 FPU, а також використовувати такі модулі у проектах C++.

Завдання:

1. Створити проект Visual C++ Win32 з ім'ям Lab11.

2. Написати на асемблері процедуру обчислення скалярного добутку двох векторів із використанням команд SSE. Ім'я процедури: **MyDotProduct_SSE.** Процедуру оформити у окремому модулі і записати файли vectsse.asm, vectsse.h. Додати файл vectsse.asm у проект.

3. Запрограмувати на асемблері процедуру обчислення скалярного добутку двох векторів на основі команд x87 FPU без використання команд SSE. Ім'я процедури: **MyDotProduct_FPU**. Процедуру оформити у окремому модулі і записати файли vectfpu.asm, vectfpu.h. Додати файл vectfpu.asm у проект.

4. Запрограмувати на C++ обчислення скалярного добутку тих самих векторів як звичайну функцію C++ з ім'ям **MyDotProduct**, яка приймає значення двох масивів і записує результат у числову перемінну (будь-яка оптимізація при компіляції повинна бути відсутня).

5. Зробити меню для вікна програми так, щоб користувач програми мав можливість викликати процедури на асемблері MyDotProduct_SSE, MyDotProduct_FPU з модулів vectsse, vectfpu, а також функцію MyDotProduct.

6. Запрограмувати вивід результатів обчислень та виміри часу виконання скалярного добутку для трьох варіантів реалізації.

7. Отримати дизасемблерний текст функції C++ MyDotProduct. Проаналізувати код дизасемблеру, порівняти з кодом на асемблері процедури MyDotProduct_FPU.

8. Зробити висновки щодо використання модулів на асемблері у програмах на мові C++ .

Теоретичні відомості

Скалярний добуток векторів

Для *n*-вимірних векторів

$\boldsymbol{A} = \{a_0, a_1, \ldots, a_{n-1}\}$

та

$\boldsymbol{B} = \{b_0, b_1, \ldots, b_{n-1}\}$

скалярним добутком є число, яке дорівнює сумі попарних добутків елементів:

$x = a_0\, b_0 + a_1\, b_1 + \ldots + a_{n-1}\, b_{n-1}$

Обчислення скалярного добутку векторів є базовою операцією для багатьох галузей – матричних обчислень, фільтрації, кореляційного аналізу тощо. Для обробки векторів зручно використовувати векторні команди SSE, AVX.

Векторні команди SSE

Ці команди виконують однакову операцію одразу над чотирма парами чисел в 32-бітовому форматі з плаваючою точкою. Наприклад:

```
addps A, B    ;додавання чотирьох чисел
subps A, B    ;додавання чотирьох чисел
mulps A, B    ;множення чотирьох чисел
divps A, B    ;ділення
rcpps A, B    ;зворотна величина
sqrtps A, B   ;квадратний корінь
rsqrtps A, B  ; 1/квадратний корінь
maxps         ;максимум
minps         ;мінімум
```

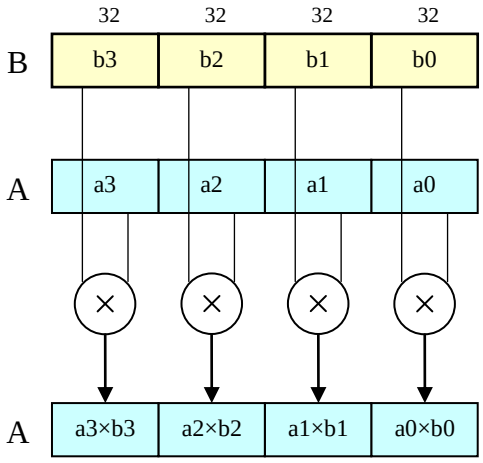

Схема виконання команди **`mulps A, B`**

Рис. 1. Приклади векторних команд SSE

Для обчислення скалярного добутку *n*-вимірних векторів можна скористатися командою **`mulps`**, яка перемножує одночасно чотири пари чисел – елементів векторів. Для зручності організації циклу обчислень потрібно, щоб *n* було кратним 4.

Накопичення сум добутків можна виконувати командою **`addps`**. Ця команда додає одночасно чотири пари чисел і зберігає чотири результати у відповідні

частини якогось регістру XMM. Так накопичуються суми добутків у вигляді чотирьох чисел у регістрі, наприклад, XMM0.

Горизонтальне додавання

Для отримання кінцевого результату скалярного добутку потрібно обчислити суму чотирьох елементів регістру XMM0, тобто виконати так зване "горизонтальне" додавання (Рис. 2):

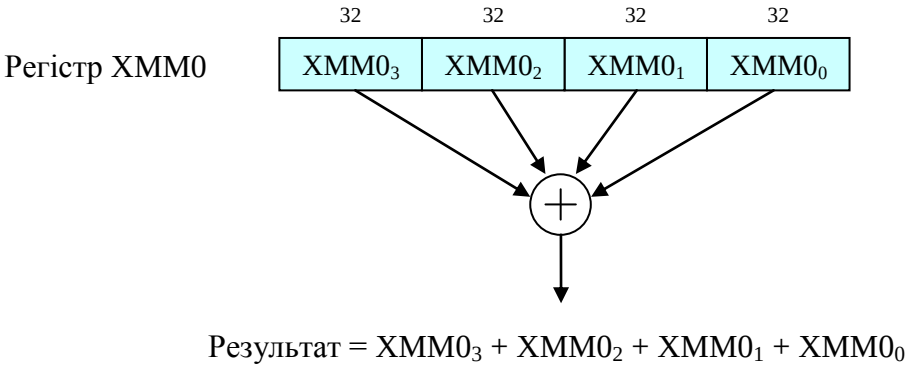

Рис. 2. "Горизонтальне" додавання елементів регістру

Знаходження суми усіх чвертинок регістру XMM0 можна виконати двома командами **haddps**:

```
haddps xmm0, xmm0
haddps xmm0, xmm0
```

Одна команда **haddps** працює так:

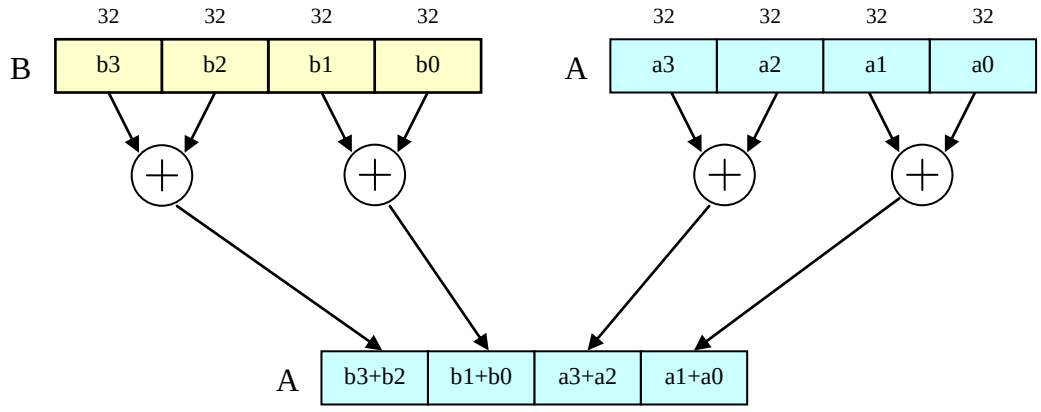

Рис. 3. Схема виконання команди **haddps A,B**

Команда **haddps** належіть до складу команд SSE3, тому якщо процесор не підтримує це розширення, то горизонтальне додавання ускладнюється – його можна запрограмувати на основі команд **shufps**.

Команда SHUFPS

Ця команда призначена для перестановки 32-бітних чвертинок у квадрослові.

```
shufps A, B, маска
```

Означає, що чвертинки операндів A та B будуть розташовані у операнді A, причому на місце двох молодших чвертинок в A будуть записані якісь чвертинки з A, а на місце двох старших чвертинок будуть записані якісь чвертинки від B.

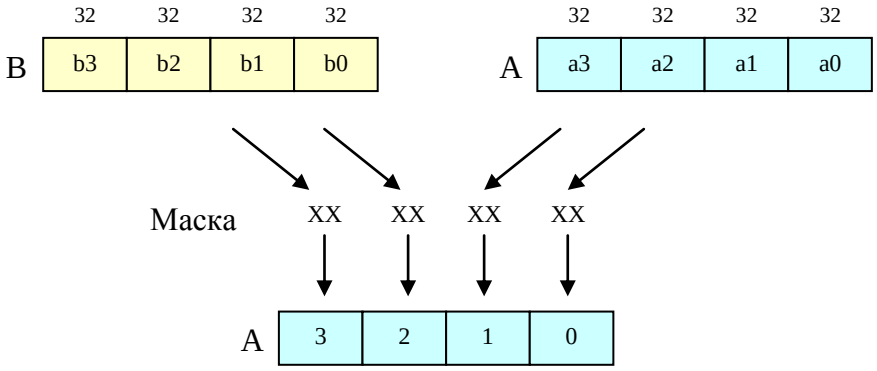

Рис. 4. Схема виконання команди **shufps**

Пари бітів 8-бітової маски вказують, які 32-бітові частини та як будуть розташовані у квадрослові A. Наприклад, маска 01101100

```
shufps A, B, 01101100b
```

означатиме наступне (рис. 5):

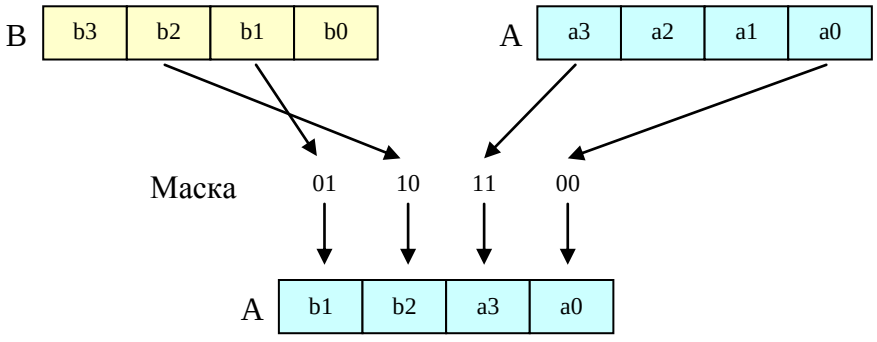

Рис. 5. Приклад роботи команди **shufps**

Вирівнювання даних

Деякі команди SSE, які зчитують дані з пам'яті, коректно працюють лише у випадку, якщо ці дані у пам'яті вирівнені на межу 16 байтів. Те саме стосується

і запису даних у пам'ять за деякою адресою. Іншими словами, адреса джерела та адреса призначення повинна бути кратною 16. Наприклад, командою

```
movaps xmm1, [ebx]
```

записується квадрослово (16 байтів) у регістр XMM1 з пам'яті за адресою, яка міститься у регістрі EBX – ця адреса повинна бути кратною 16, інакше виконання програми буде аварійно завершене.

На відміну цього, командою

```
movups xmm1, [ebx]
```

можна зробити те саме і для невирівнених даних, проте команда **movups** працює суттєво повільніше, аніж **movaps**. Це обов'язково потрібно враховувати. Якщо продедура на асемблері обробляє дані, запрограмовані у модулях на C++, то ці дані повинні бути вирівнені. Запрограмувати на C++ створення статичного масиву, адреса якого кратна 16, можна так:

```
__declspec( align(16) ) float a[1000];
```

Програмування скалярного добутку на основі команд x87 FPU

Для обчислення скалярного добутку можна використати такі команди x87 FPU:

FADD операнд

– виконується додавання операнду до ST(0) і запис результату у ST(0).

FMUL операнд

– виконується множення операнду на ST(0) і запис результату у ST(0).

Примітка. Є також інші різновиди цих команд (див. у документації Intel® 64 and IA-32 Architectures. Software Developer's Manual).

Для виконання операцій потрібно завантажувати дані у стек FPU. Вершина стеку, яка зветься ST(0), спочатку має адресу 0 (вказує на регістр R0 стеку з восьми регістрів R0-R7). Команда FLD завантажує одне число у стек – спочатку зменшується на одиницю адреса вершини стеку ST(0), а потім у ST(0) записується значення операнду. Докладні відомості щодо цього викладені у розділі теоретичних відомостей попередньої лабораторної роботи №8.

Приклад обчислення суми двох пар добутків Res = A×B + C×D

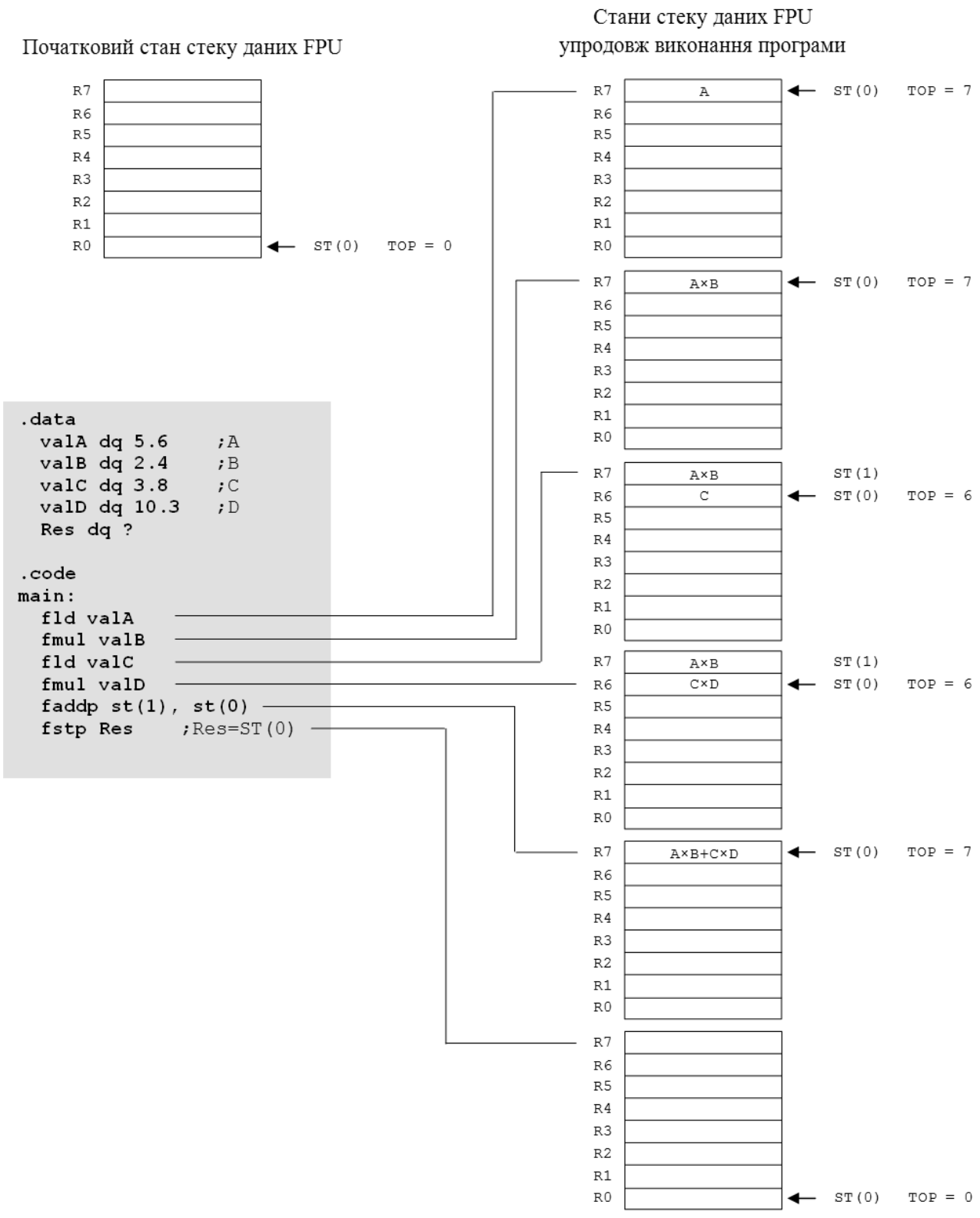

Рис. 6. Приклад обчислення двох пар добутків

Оскільки для виконання завдання даної лабораторною потрібно додавати багато пар добутків, то необхідно запрограмувати цикл, у якому будуть виконуватися команди множення та додавання чисел з плаваючою точкою.

Порядок виконання роботи та методичні рекомендації

1. У середовищі MS Visual Studio створіть новий проект C++ Win32 з ім'ям **Lab11.** Після створення проекту виконайте настроювання проекту для підтримки асемблеру. Основні пояснення щодо цього містяться у відповідних розділах попередньої лабораторної роботи №10. Скомпілюйте початковий зразок програми, автоматично сгенерований середовищем Visual Studio. Після успішного виклику програми можна продовжувати наступні кроки.

2. Додайте у проект модуль на асемблері – файл **vectsse.asm**. Напишіть вихідний текст модуля згідно завданню. Для того, щоб було можливим використовувати у модулі команди SSE, текст модуля повинен розпочинатися такими рядками:

```
.686
.xmm
.model flat, C
```

3. Додайте у проект ще один модуль на асемблері – файл **vectfpu.asm.** Цей модуль повинен містити програмний код процедури обчислення скалярного добутку на основі команд x87 FPU згідно завданню.

4. Скомпілюйте модулі окремо. Після налагодження модулів, напишіть вихідний текст головного файлу, який буде містити наступне:

```
#include "vectsse.h"
#include "vectfpu.h"
. . .                                   //оголошення масивів даних A[N], B[N]
. . .                                   //інший програмний код
MyDotProduct_SSE(&res, A, B, N);        //виклик функції - процедури на асемблері
. . .                                   //інший програмний код
MyDotProduct_FPU(&res, A, B, N);        //виклик функції - процедури на асемблері
. . .                                   //інший програмний код
MyDotProduct(&res, A, B, N);            //виклик функції на C++
. . .                                   //вивід результатів
```

У головному файлі присутні виклики трьох функцій, вони обчислюють той самий скалярний добуток для того, щоб порівняти їхню швидкодію.

5. Запрограмуйте знаходження часу виконання для кожного з варіантів обчислення скалярного добутку. Як запрограмувати знаходження часу виконання потрібних функцій? Одним з варіантів рішення є використання функції API Win32 GetLocalTime, наприклад, у такий спосіб:

```
SYSTEMTIME st;
long tst,ten;
GetLocalTime(&st);
tst = 60000*(long)st.wMinute
    + 1000*(long)st.wSecond
    + (long)st.wMilliseconds;
for (long i=0; i<1000000; i++)     //повторюємо мільйон разів
    {
    . . .          //код, для якого потрібно виміряти час
    }
GetLocalTime(&st);
ten = 60000*(long)st.wMinute
    + 1000*(long)st.wSecond
    + (long)st.wMilliseconds - tst;
```

Для коректного виміру часу потрібно встановити таку кількість повторень, щоб час виконання циклу був не менше секунди.

Програма повинна виводити числове значення скалярного добутку та час виконання у вікні MessageBox окремо для кожного з варіантів реалізації.

6. Налагодження програми та отримання результатів. Налагодьте програму та запишіть отримані числові значення скалярного добутку та часу виконання відповідних операцій.

7. Отримайте дизасемблерний код процедури MyDotProduct на C++ і порівняйте з кодом процедури на асемблері MyDotProduct_FPU. Проаналізуйте відмінності програмних кодів, якщо вони є.

Варіанти завдання

Кількість елементів векторів A та B має бути N=40*(номер студента у журналі)

Зміст звіту:

1. Титульний лист
2. Завдання
3. Роздруківка вихідного тексту програми:
   - вихідних текстів модуля на асемблері – повністю
   - головного файлу lab11.cpp – частково, тільки текст, який був доданий для виконання завдання
   - дизасемблерний код функції **MyDotProduct** на C++
4. Роздруківка результатів виконання програми
5. Аналіз, коментар результатів та вихідного тексту
6. Висновки

Контрольні питання:

1. Як працюють команди SSE?
2. Що таке "горизонтальне" додавання і яке тоді додавання "вертикальне"?
3. Що таке вирівнювання даних і як воно вказується?
4. Що таке стек даних FPU?
5. Що потрібно вказати, щоб успішно компілювався вихідний текст на асемблері з використанням команд SSE?
6. Як можна запрограмувати виміри часу виконання потрібних операцій?